2. Beadandó feladat (Négyzetek) dokumentáció

Készítette:

Vörös Döme

Neptun-Kód: QK8IUC

E-mail: qk8iuc@inf.elte.hu

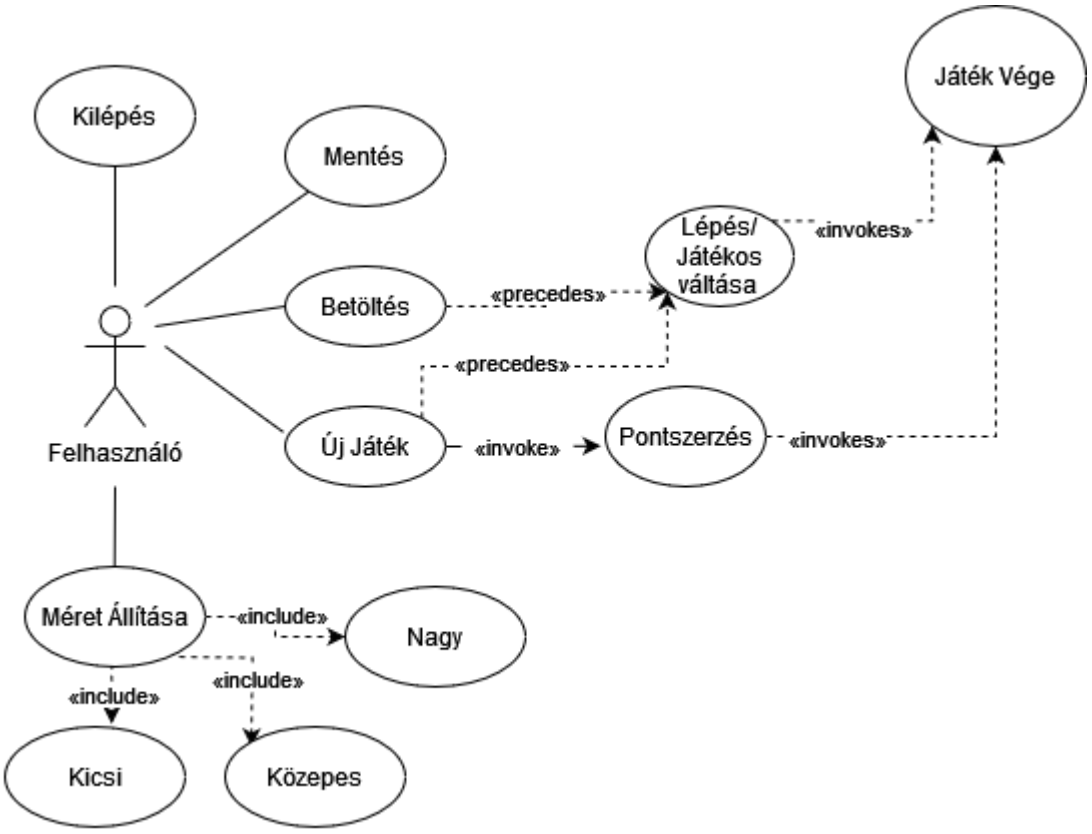
Feladat:

Készítsünk programot, amellyel az alábbi két személyes játékot játszhatjuk. Adott egy $n \times n$ pontból álló játéktábla, amelyen a játékosok két szomszédos pont között vonalakat húzhatnak (vízszintesen, vagy függőlegesen). A játék célja, hogy a játékosok a húzogatással négyzetet tudjanak rajzolni (azaz ők húzzák be a negyedik vonalat, független attól, hogy az eddigieket melyikük húzta). Ilyen módon egyszerre akár két négyzet is elkészülhet. A játék addig tart, amíg lehet húzni vonalat a táblán. A játékosok felváltva húzhatnak egy-egy vonalat, de ha egy játékos berajzolt egy négyzetet, akkor ismét ő következik. A program biztosítson lehetőséget új játék kezdésére a pályaméret megadásával (3×3, 5×5, 9×9), játék mentésére és betöltésére. Ismerje fel, ha vége a játéknak, és jelenítse meg, melyik játékos győzött (ha nem döntetlen). Játék közben a vonalakat, illetve a négyzeteket színezze a játékos színére.

Elemzés:

- A játékot három nehézségi szinttel játszhatjuk: könnyű (3x3as tábla), közepes (5x5ös tábla), nehéz (9x9es tábla). A program indításkor könnyű szintet állít be, és automatikusan egy új játékot indít el.
- A feladatot egyablakos asztali alkalmazásként Windows Presentation Foundation grafikus felülettel valósítjuk meg.
- Az ablakban elhelyezünk egy menüt a következő menüpontokkal: File (New Game [Új Játék], Load Game [Játék Betöltése], Save Game [Játék Mentése], Exit Game [Kilépés]). Az ablak alján megjelenítünk egy státuszsort, amely az aktuális játékost és a két játékos pontszámát írja ki.
- A játéktáblát egy pályamérettől függő (7x7, 11x11, 19x19) nyomógombokból álló rács reprezentálja. A pályán szerepelnek fekete négyzetek, amelyek a nem használható mezőket jelentik, amíg vannak a kattintható vonalakat reprezentáló gombok, és a négyzetek, amelyek nem kattinthatóak de színt váltanak ha a vonalak körülöttük színesek lesznek.
- A játék automatikusan feldob egy dialógusablakot, amikor vége a játéknak (nyert valamelyik játékos vagy döntetlen lett) vagy amikor ki akarunk a programból lépni. Szintén dialógusablakokkal végezzük el a mentést, illetve betöltést, a fájlneveket a felhasználó adja meg.

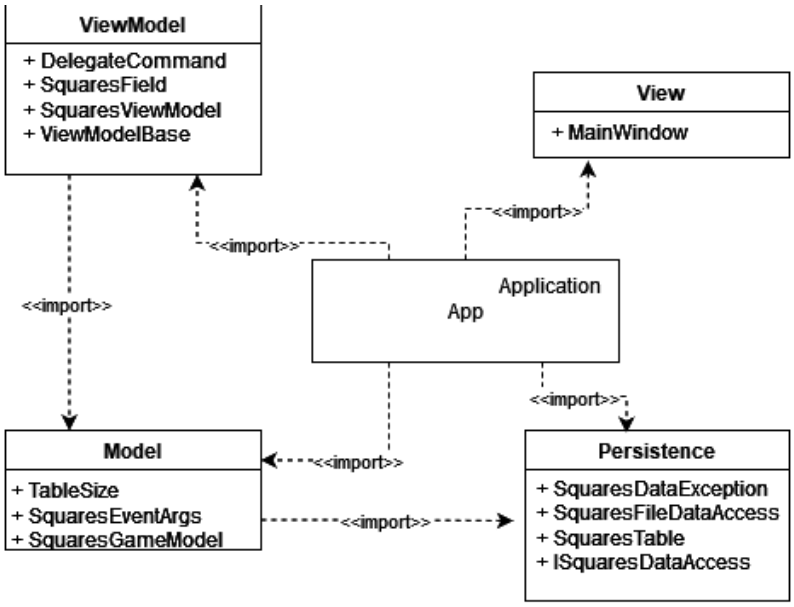
- A felhasználói esetek az 1. ábrán láthatóak.



1. ábra: Felhasználói esetek diagramja

Tervezés:

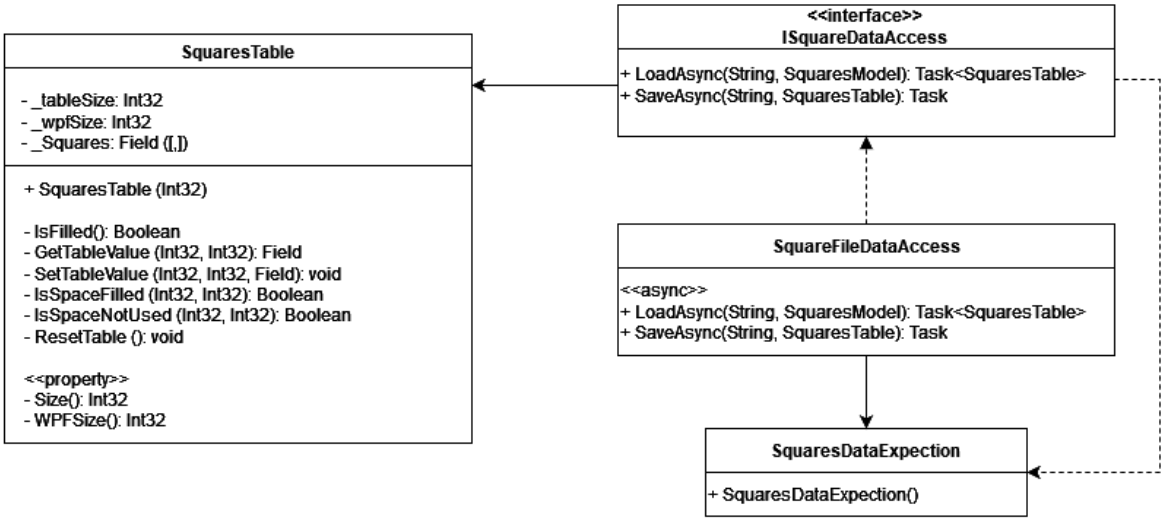
- Programszerkezet:
  - A programot MVVM architektúrában valósítjuk meg. A megjelenítés a View és ViewModel, a modell a Model, míg a perzisztencia Persistence névtérben helyezkedik el. A program környezetét az alkalmazás osztály (App) végzi, amely példányosítja a modellt, a nézetmodellt és a nézetet, biztosítja a kommunikációt, valamit felügyeli az adatkezelést. A program csomagszerkezete a 2. ábrán látható.
  - A program szerkezetét két projektre osztjuk implementációs megfontolásból: a Persistence és Model csomagok a program felületfüggetlen projektjében, míg a ViewModel és View csomagok a WPF függő projektjében kapnak helyet.



2. ábra: Az alkalmazás csomagdiagramja

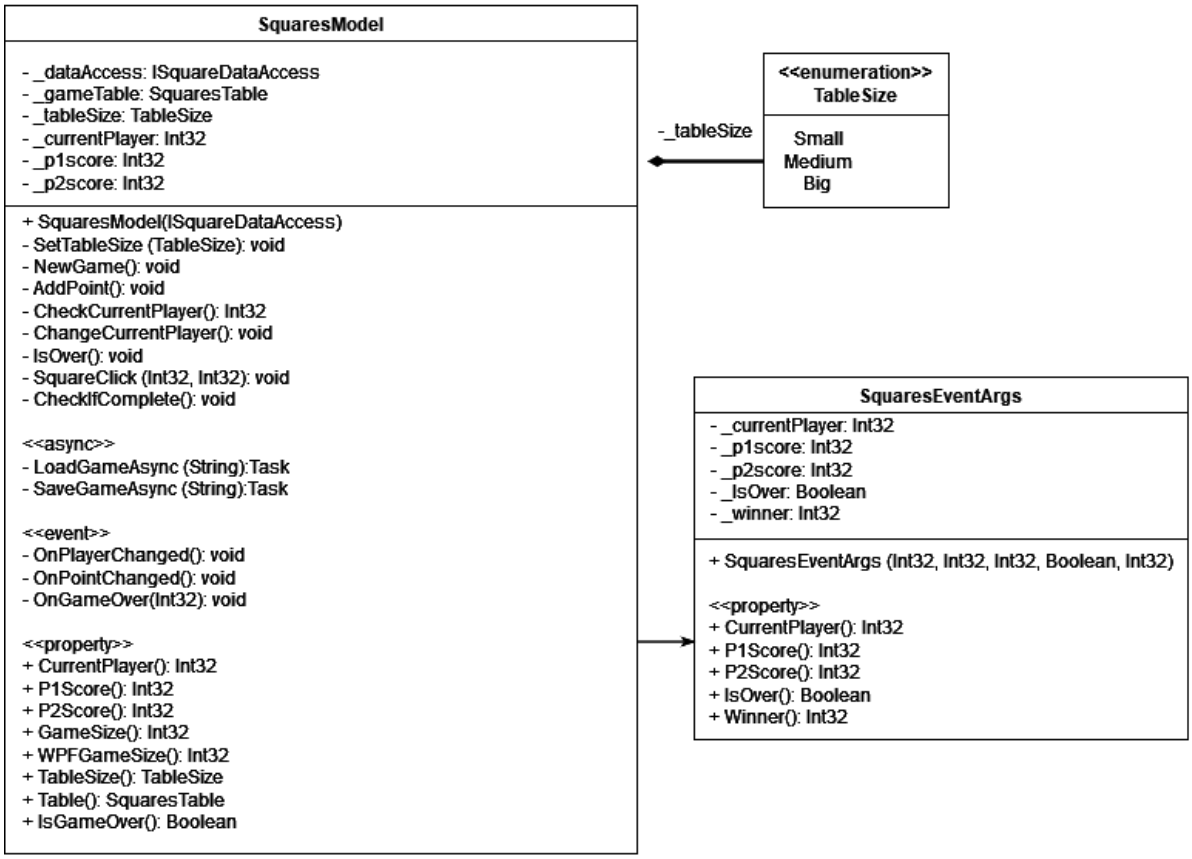
- Perzisztencia:
  - Az adatkezelés feladata a Squares táblával kapcsolatos információk tárolása, valamint a betöltés/mentés biztosítása.
  - A **SquaresTable** osztály egy érvényes Squares táblát biztosít (azaz mindig ellenőrzi a beállított értékek), ahol a mezőcsoportnak ismert az Field (**NotUsed**, **Empty**, **Player1**, **Player2**) értéke. A játék kezdetekor mindegyik mező **Empty vagy NotUsed** értéket fog kapni. A tábla alapértelmezés szerint 7 x 7 -es alapmezőkkel rendelkezik, de ez a konstruktorban paraméterezhető. A tábla lehetőséget az állapotok lekérdezésére (**IsFilled**, **GetTableValue**, **IsSpaceFilled, IsSpaceNotUsed**), direkt beállítás (**SetTableValue**) elvégzésére, illetve a tábla elemeinek újra beállítására, amelyet egy játék indításakor alkalmazunk (**ResetTable**).
  - A hosszú távú adattárolás lehetőségeit az **ISquaresDataAccess** interfész adja meg, amely lehetőség ad a tábla betöltésére (LoadAsync), valamint mentésére (SaveAsync). A műveleteket hatékonysági okokból aszinkron módon valósítjuk meg
  - Az interfészt szöveges fájl alapú adatkezelésre a **SquaresFileDataAccess** osztály valósítja meg. A fájlkezelés során fellépő hibákat a **SquaresDataExpection** kivétel jelzi.
  - A program az adatokat szöveges fájlként tudja eltárolni, melyek az **sqr** kiterjesztést kapják. Ezeket az adatokat a programban bármikor be lehet tölteni, illetve ki lehet menteni az aktuális állást.
  - A fájl első sora megadja a tábla méretét, az első és második játékos pontszámát, illetve az aktuális játékos számát (1 – első játékos, 2 – második játékos). A fájl többi része izomorf leképzése a játéktábla 3 különböző részének azaz kis tábla esetén 7 sor, a közepes tábla esetén 11 sor, a nagy tábla esetén 19 sor következik, melyben az aktuális mezőnek van leírva a Field állapota (nem használt, üres, első játékosé, második játékosé). A

számok 0-2 lehetnek, ahol a 0 az üres mezőt, az 1 a első játékos mezőjét, a 2 pedig a második játékos mezőjét adja meg.



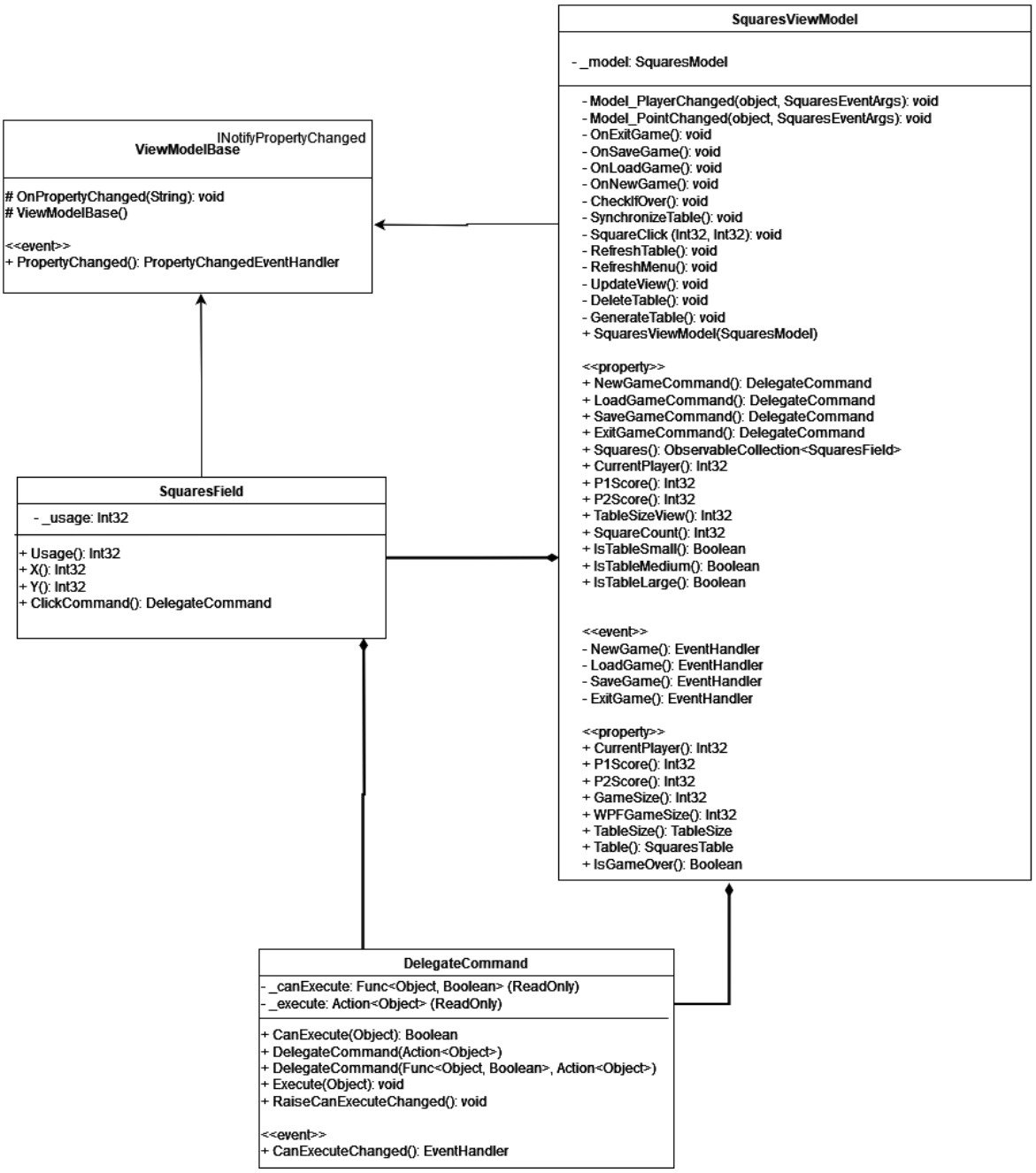
3. ábra: A Persistence csomag osztálydiagramja

- Modell:
  - A modell lényegi részét a **SquaresGameModel** osztály valósítja meg, amely szabályozza a tábla tevékenységeit, valamit a játék egyéb paramétereit, úgymint a két játékos pontszáma (**_p1Score**, **_p2Score**) és az aktuális játékost (**_currentPlayer**). A típus lehetőséget ad új játék kezdésére (**NewGame**), pontszám hozzáadáshoz (**AddPoint**), aktuális játékos lekérdezésére (**CheckCurrentPlayer**) és megváltoztatására (**ChangeCurrentPlayer**), a játék végének megtörténésére (**IsOver**) és egy adott négyzet ellenőrzésére, hogy be lehet-e színezni (**CheckIfComplete**)
  - A játékállapot változásáról (pontszám változás, játékos változás) a **PointChanged** és **PlayerChanged** esemény, míg a játék végéről a **GameOver** esemény tájékoztat. Az események argumentuma (**SquaresEventArgs**) tárolja a győzelem állapotát, a pontszámokat, és az aktuális játékost.
  - A modell példányosításkor megkapja az adatkezelés felületét, amelynek segítségével lehetőséget ad betöltésre (**LoadGameAsync**) és mentésre (**SaveGameAsync**)
  - A játéktábla méretét a **TableSize** felsorolási típuson át kezeljük, és a **SquaresGame** osztályban konstansok segítségével tároljuk az egyes nehézségek paramétereit.



4. ábra: A Model csomag osztálydiagramja

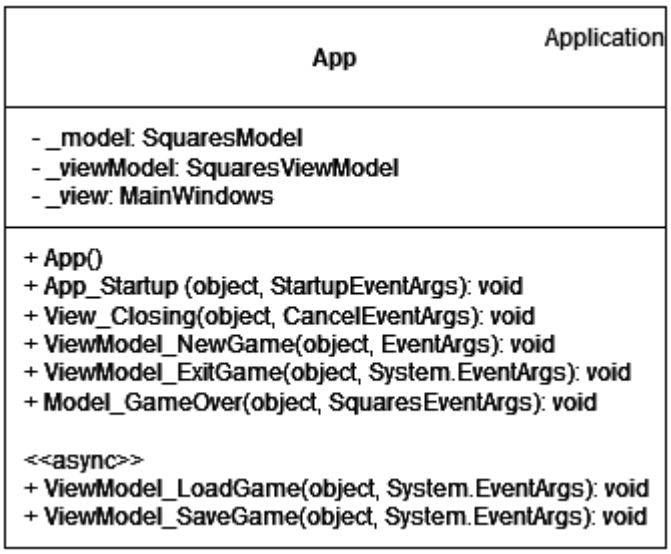
- Nézetmodell:
  - A nézetmodell megvalósításához felhasználunk egy általános utasítás (**DelegateCommand**), valamint egy ős változásjelző (**ViewModelBase**) osztályt.
  - A nézetmodell feladatait a **SquaresViewModel** osztály látja el, amely parancsokat biztosít az új játék kezdéséhez, játék betöltéséhez, mentéséhez, valamint a kilépéshez. A parancsokhoz eseményeket kötünk, amelyek a parancs lefutását jelzik a vezérlőnek. A nézetmodell tárolja a modell egy hivatkozását (**_model**), de csupán információkat kér le tőle, illetve a játéknehézséget szabályozza. Direkt nem avatkozik a játék futásába.
  - A játékmező számára egy külön mezőt biztosítunk (**SquaresField**), amely eltárolja a pozicíót, használati módját, valamit a kattintás parancsát (**ClickCommand**). A mezőket egy felügyelt gyűjteménybe helyezzük a nézetmodellbe (**Fields**).



5. ábra: A nézetmodell osztálydiagramja

- Nézet:
  - A nézet csak egy képernyőt tartalmaz, a MainWindow osztályt. A nézet egy rácsban tárolja a játékmezőt, a menüt és a státuszsort. A játékmező egy **ItemsControl** vezérlő, ahol dinamikusan felépítünk egy rácsot (UniformGrid), amely gombokból áll. Minden adatot adatkötéssel kapcsolunk a felülethez, továbbá azon keresztül szabályozzuk a gombok színét is.
  - A fájlnév bekérését betöltéskor és mentéskor, valamint a figyelmeztető üzenetek megjelenítését beépített dialógusablakok segítségével végezzük.

- Környezet:
  - Az App osztály feladata az egyes rétegek példányosítása (**App_Startup**), összekötése, a nézetmodell, valamint a modell eseményeinek lekezelése, és ezáltal a játék, az adatkezelés, valamint a nézetek szabályozása.



6. ábra: A vezérlés osztálydiagramja

Tesztelés:

- A modell funkcionalitása egységtesztek segítségével lett ellenőrizve a **SquaresGameModelTest** osztályban.
- Az alábbi tesztesetek kerültek megvalósításra:
  - **SquaresGameModelNewGameEasyTest** **SquaresGameModelNewGameMediumTest** **SquaresGameModelNewGameHardTest**: Új játék indítása, a mezők beállítása, a két játékos pontszáma és az aktuális játékos ellenőrzése, illetve a mezők megszámolása a táblaméret függvényében.
  - **SquaresGameModelPlayersTest**: Új táblán mindkét játékos lép egyet, mely közben ellenőrzöm, hogy nem ad nekik pontszámot és hogy megfelelően váltogatja az aktuális játékost.
  - **SquaresGameModelPointTest**: Új táblán beállítok pont annyi mezőt, hogy a következőnél teljes legyen a négyzet és a lépés előtt és után is tesztelem hogy a pontszámok megfelelően növekednek és hogy a teljes négyzet miatt az aktuális játékos mégegyszer léphet majd.
  - **SquaresGameModelTwoPointTest**: Új táblán beállítok pont annyi mezőt, hogy a következőnél kettő négyzet is teljes legyen és a lépés előtt és után is tesztelem hogy a pontszám 2-vel növekszik és hogy a teljes négyzet miatt az aktuális játékos mégegyszer léphet majd.